Implementing Real Root Count with Polynomial Constraints for the Multivariate Case with one Constraint Polynomial

Hoon Hong
Wolfgang Stöcher
Research Institute for Symbolic Computation
Johannes Kepler University, 4040 Linz, Austria

December 1, 1995

Abstract

Following [P. Pederson, et al. 94] and based on [H. Hong et al. 95] we develop several algorithms to compute multiplication tables and associated quadratic forms, which we compare in terms of efficiency. The results can immediately be applied to the general case of arbitrary many polynomials.

1 Introduction

Problem specification

We will use $\bar{x}$ as an abbreviation for $x_1, \ldots x_r$ and let K denote any subfield of a real closed field R. Then we consider the following problem:

given: A set of polynomials $\{f_1, \ldots f_n\}$
such that $f_i \in K[\bar{x}]$ and the system $f_1 = \ldots = f_n = 0$ has finitely many solutions,
and a constraint polynomial $h \in K[\bar{x}]$.

find: The number of common real roots of the polynomials f_i
on which the constraint $h > 0$ holds.

Since $h \geq 0$ is equivalent to $(h = 0 \vee h > 0)$, we can reduce any polynomial constraint to the specified problem.

Theory

Similar to [P. Pederson, et al. 94] we introduce the following notations. Consider the finite dimensional vector space $A = K[\bar{x}]/I$. We define:

- $m_f : A \to A$, $g \mapsto m_f(g) = fg$.
- $B : A \times A \to K$, $(f, g) \mapsto B(f, g) = \mathrm{Tr}(m_f m_g) = \mathrm{Tr}(m_{fg})$, where Tr is the trace function.
- $B^{<h>} : A \times A \to K$, $(f, g) \mapsto B(hf, g) = \mathrm{Tr}(m_{fgh})$.

In a given Basis V for A the matrices for B and $B^{<h>}$ shall be denoted by Q and $Q^{<h>}$, respectively. $V_R(I)$ shall denote the real affine variety of the ideal I. With this notation the main theorem can be stated like this:

Theorem 1.1 *Given a field K, a finite dimensional ideal I, generated by $F = \{f_1, \ldots f_n\}$, where $f_i \in K[\bar{x}]$, and given R, a real closed field such that $K \subset R$, and given another polynomial $h \in K[\bar{x}]$, then*

$$\sigma(Q^{<h>}) = \sharp\{\bar{x} \in V_R(I) : h(\bar{x}) > 0\} - \sharp\{\bar{x} \in V_R(I) : h(\bar{x}) < 0\}. \quad (1)$$

Hence we have

$$\sigma(Q^{<h^2>}) = \sharp\{\bar{x} \in V_R(I) : h^2(\bar{x}) > 0\} + 0 = \sharp\{\bar{x} \in V_R(I) : h(\bar{x}) > 0\} + \sharp\{\bar{x} \in V_R(I) : h(\bar{x}) < 0\}.$$

and the desired number of common real roots fulfilling a given constraint h is given by

$$c_{>0}(F, h) := \sharp\{\bar{x} \in V_R(I) : h(\bar{x}) > 0\} = \frac{1}{2}(\sigma(Q^{<h>}) + \sigma(Q^{<h^2>})). \tag{2}$$

2 Representation

Basis

For efficient computation in $A = K[\bar{x}]/I$ we represent the equivalence classes as vectors with respect to a certain basis. Since I is 0-dimensional the basis is finite. The following theorem allows us to choose the basis elements ??
as power products:

Theorem 2.1 *Let $I \subset K$ be an ideal. Let $<LT(I)>$ denote the ideal generated by the leading terms in I and $\bar{x}^\alpha$ shall be an abbreviation for $x_1^{a_1} \dots x_r^{a_r}$. Then $K[\bar{x}]/I$ is isomorphic as a K-vector space to $S = \mathrm{Span}(\bar{x}^\alpha : \bar{x}^\alpha \notin <LT(I)>)$.*

Let us denote such a basis for A by $V = \{v_1, \dots v_d\}$ and the representation of an element $f \in A$ shall be denoted by $f^* = (f^1, \dots f^d)^T$, i. e. using tensor notation $f = f^k v_k$.

Multiplication Table

For multiplying the representation of two elements of A it is sufficient to know how two basis elements are multiplied. This information we keep in the multiplication table w, a square matrix of vectors with the same dimension:

$$w_{ij}^k := (v_i v_j)^k, \qquad 1 \leq i, j, k \leq d \tag{3}$$

which is equivalent to (we will use tensor notation throughout) $v_i v_j = w_{ij}^k v_k$. For more details on the structure of a Multiplication Table see [H. Hong et al. 95].

Associated Quadratic Forms

Since $B(f, g) = B^{<1>}(f, g)$ we can restrict ourself to computing the matrix $Q^{<h>}$. In [H. Hong et al. 95] the following formula is derived:

$$Q_{ij}^{<h>} = w_{ij}^k \, \mathrm{Tr}(m_{hv_k}^*), \tag{4}$$

such that

$$B^{<h>}(f, g) = (f^*)^T \, Q^{<h>} \, g^*.$$

3 Outline of Implementation

Suppose we know the Groebner basis GB of the ideal $I := <f_1, \dots f_n>$ generated by the polynomials of the input, the algorithm proceeds in the following steps:

1. Construct a basis $V = \{v_1, \dots v_d\}$ of $A = K[\bar{x}]/I$.
2. Compute the multiplication table w_{ij}^k (regarded as the matrix w_{ij} of vectors indexed by k).
3. Compute the associated quadratic forms $Q, Q^{<h>}$ and $Q^{<h^2>}$.
4. Compute signatures $\sigma(Q^{<h>})$, $\sigma(Q^{<h^2>})$.
5. The arithmetic mean of the two signatures $\frac{1}{2}(\sigma(Q^{<h>}) + \sigma(Q^{<h^2>}))$ gives the number of common real roots fulfilling the constraint h, i. e. $h > 0$.

Table 1: Timings for computing the multiplication table w

Data	#(all pairs)	#(dist. pairs)	ratio	comp. all pairs	comp. dist. pairs	speedup
4 var/deg 2	136	31	4.39	26.8 s	8 s	3.35
4 var/deg 2	136	45	3.02	41.2 s	20.6 s	2.0
3 var/deg 3	105	63	1.67	8.6 s	5.3 s	1.62

4 Efficient Computation of a Multiplication Table

By theorem 2.1 a basis can be built of all power products not in $LT(I)$, usually referred to as the *Groebner staircase*. Those power products are generated in lexicographical order ($x_1 < x_2 < \ldots$): $v_1 < v_2 < \ldots < v_d$ w.r.t. this ordering (hence $v_1 = 1$ and $v_i < v_j$ is equivalent to $i < j$).

For fixed i, j the vector w_{ij}^k, $1 \leq k \leq r$, is computed by reducing the product $v_i v_j$ with the Groebner basis GB to its normal form. By the construction of V the normal form is a linear combination of the basis elements $v_k, 1 \leq k \leq r$, whose coefficients are extracted into w_{ij}^k.

To avoid multiple reduction of the same polynomial and to keep the structural information for later use in the *structure matrix* Qs we proceed as follows (see [H. Hong et al. 95] for more details on the following algorithm):

$DistCtr := 0$
for $i := 1$ to r do
  for $j := i$ to r do
    if w_{ij} is not already computed
      $DistCtr := DistCtr + 1$
      $Qs_{ij} := Qs_{ji} := DistCtr$
      $p := v_i\, v_j$
      reduce p to its normal form n w.r.t. GB (i. e. the basis representation w.r.t. V)
      extract the coefficients of n (w.r.t. basis V) as w_{ij}^k and w_{ji}^k.
      for all pairs (l, m), $i < l \leq m < j$, do
        if $(v_l v_m = p)$
          $Qs_{lm} := Qs_{ml} := DistCtr$
          $w_{lm}^k := w_{ml}^k := w_{ij}^k, 1 \leq k \leq d$.

The gain heavily depends on the basis, more precisely on the structure of the Groebner staircase. For polynomials in two variables we shall give an idea how much computation is saved. Let us denote the set of all pairwise combinations of basis elements by V^2, from which we want to estimate the cardinality in terms of the number of basis elements $|V| = d$. The number of basis elements is roughly the area covered by the Groebner staircase. Now building all pairwise products corresponds to adding all pairs of exponent vectors of the Groebner staircase, which roughly increases the covered area by 4 (2 in each of the 2 directions corresponding to the exponent of one variable). Hence from geometric intuition we estimate $|V^2| \approx 4|V| = 4d$. Thus we get an estimate of the speedup by

$$\frac{\#(\text{all pairs})}{\#(\text{distinct pairs})} \approx \frac{d(d+1)}{2 \cdot 4d} = \frac{d+1}{8}.$$

This is in general a too restrictive bound, as one can see from examples in table 1. Another way to see the gain and how it depends on the structure of the Groebner staircase is by directly looking at a product of two basis elements and count the number of pairs giving the same product. Let us stick again to the case of two variables and consider a product $v_1 v_i = 1 \cdot x^{i_x} y^{i_y} = x \cdot x^{i_x - 1} y^{i_y} = v_2 v_{i-1} = \cdots$. In the Groebner staircase the basis elements involved with these different decompositions of the product build a rectangle with two opposite corners being the exponent vectors of v_1 and v_i. Hence we have $(i_x + 1)(i_y + 1)/2$ pairs of basis elements with equal product.

The worst case of all pairwise products to be distinct can only happen with a basis being a subset of $\{1, x_1, x_2, \ldots\}$, since having at least the second power of one variable in the basis will cause at least two equal products.

Recursive Computation

Consider $v_i = v_{i_1} v_{i_2}$ such that $1 < i_1 \leq i_2 < i$. Then $v_i v_j = v_{i_1}(v_{i_2} v_j)$ and hence we have:

$$w_{ij}^k = w_{i_1 l}^k w_{i_2 j}^l.$$

By the above described order in computing the multiplication table w, the entries $w_{i_1 l}^k, 1 \leq l \leq d$, and $w_{i_2 j}^k$ will already be computed when coming to w_{ij}^k. This gives the following modified algorithm:

```
DistCtr := 0
for i := 1 to d do
    reducible := false
    if there exist 1 < i_1 ≤ i_2, s. t. v_i = v_{i_1} v_{i_2}
      reducible := true
    for j := i to d do
      if w_{ij} is not already computed
        DistCtr := DistCtr + 1
        Qs_{ij} := Qs_{ji} := DistCtr
        p := v_i v_j
        if (reducible)
          for k := 1 to d do
              compute w^k_{ij} = w^k_{ji} by summing up w^k_{i_1 l} w^l_{i_2 j}
        else
          reduce p to its normal form n w.r.t. GB (i. e. the basis representation w.r.t. V)
          extract the coefficients of n (w.r.t. basis V) as w^k_{ij} and w^k_{ji}.
        for all pairs (l, m) with v_l v_m = p
            Qs_{lm} := Qs_{ml} := DistCtr
            w^k_{lm} := w^k_{ml} := w^k_{ij}, 1 ≤ k ≤ d.
```

Incremental Computation

This time we store the computed normal forms and make use of them later on. Again we take advantage of decomposable basis elements $v_i = v_{i_1} v_{i_2}$ such that $1 < i_1 \leq i_2 < i$. Hence instead of reducing $v_i v_j$ we reduce $v_{i_1} n_{i_2 j}$, where $n_{i_2 j}$ is the normal form of $v_{i_2} v_j$. Thus we have to compute the normal form of a polynomial of lower degree and reduction runs faster. The implementation is straight forward. A modified version for computation over integers is given below.

Computation over Integers

Multiplications are a lot more efficient when done over the integers than when computing with rational numbers. But although when given integral polynomials the normal form w.r.t. a Groebner basis need not be integral. Hence more care has to be taken.

The above given algorithms only work over integers if the Groebner basis is *monic*, i. e. the leading coefficients of the polynomials are ± 1. In this case the normal form of integral polynomial will always be integral again. For a general Groebner basis the above algorithms have to be modified. Polynomials are reduced to an integral multiple of the normal form. The lifting factor must be multiplied onto the other normal forms, in order to get a consistent multiplication table[1]. In practice the least common multiple of all lifting factors is computed and all normal forms are lifted to the same multiple.

This method does not work well for the *incremental* computation. For the *recursive* computation the modified algorithm looks like the following.

```
lcmf := 1   // least common multiple of lifting factors
DistCtr := 0
for i := 1 to d do
```

[1] For counting real roots it does not matter if we use the actual multiplication table or a multiple of it. But having different factors multiplied onto the elements in the multiplication table changes the structure and gives wrong results.

```
    reducible := false
    if there exist 1 < i_1 ≤ i_2, s. t. v_i = v_{i_1} v_{i_2}
      reducible := true
    for j := i to d do
      if w_ij is not already computed
        DistCtr := DistCtr + 1
        Qs_ij := Qs_ji := DistCtr
        p := v_i v_j
        if (reducible)
          pr := v_{i_1} n_{i_2 j}
        else
          pr := p
        reduce pr to an integral multiple n = f_DistCtr n_DistCtr of its normal form w.r.t. GB
        if (reducible)
          RefNum := Qs_{i_2 j}
          f_DistCtr := f_DistCtr * f_RefNum
        extract the coefficients of n (w.r.t. basis V) as wDist^k_DistCtr.
        lcmf := lcmf f_DistCtr / gcd(lcm_f, f_DistCtr)
        for all pairs (l, m) with v_l v_m = p
            Qs_lm := Qs_ml := DistCtr

DistCtr := 1
for i := 1 to d do
    for j := i to d do
      if (Qs_ij = DistCtr)
        f := lcmf/f_DistCtr
        for k := 1 to d do
            wDist^k_distinc := f wDist^k_DistCtr
        w_ij := wDist_DistCtr
        DistCtr := DistCtr + 1
      else
        RefNum := Qs_ij
        w_ij := w_ji := wDist_RefNum
```

5 Efficient Computation of Quadratic Forms

Once the multiplication table is provided we can proceed to compute the associated quadratic forms. We first compute the associated quadratic form for $h = 1$, denoted by Q. We start by computing

$$t_k := \mathrm{Tr}(m^*_{v_k}) = \sum_l w^l_{kl}, \quad 1 \le k \le d.$$

Now from (4) and setting $h := 1$ we have

$$Q_{ij} = w^k_{ij}\, t_k. \tag{5}$$

(See [H. Hong et al. 95] for more details.) The information about equality of products of basis elements is already kept in the structure matrix Qs, so we can compute Q as follows:

```
for k := 1 to d do
    t_k := Σ_l w^l_kl
DistCtr := 0 // counting the number of distinct matrix elements
for i := 1 to d do
    for j := i to d do
        RefNum := Qs_ij
        if (DistCtr < RefNum)
          DistinctElements[RefNum] := Σ_k w^k_ij t_k
          DistCtr := DistCtr + 1
```

(remark: by construction of Qs we now have $DistCtr = RefNum$)
$Q_{ij} = DistinctElements[RefNum]$

Computing the quadratic forms $Q^{<h>}$ and $Q^{<h^2>}$

In [H. Hong et al. 95] a more general equation for $Q^{<h_1h_2>}$ is deduced, namely the recursion formula:

$$Q_{ij}^{<h_1h_2>} = w_{ij}^k Q_{kn}^{<h_1>} h_2^n. \tag{6}$$

Now defining for $h = h_1h_2$

$$t_k^{<h>} := Q_{kn}^{<h_1>} h_2^n$$

and having computed the basis representation of h we can proceed in the following way:

1. $t_k^{<h>} = Q_{kn}h^n$,
2. $Q_{ij}^{<h>} = w_{ij}^k t_k^{<h>}$,
3. $t_k^{<h^2>} = Q_{kn}^{<h>} h^n$,
4. $Q_{ij}^{<h^2>} = w_{ij}^k t_k^{<h>}$.

The algorithm for computing $Q^{<h>}$ and $Q^{<h^2>}$ in an analogous to the one given above for computing Q, again using the structure matrix Qs.

6 Computing the Signatures $\sigma(Q^{<h>})$ and $\sigma(Q^{<h^2>})$

To compute the signature of a symmetric matrix we transform it congruently to its diagonal form, i. e. the symmetric matrix is diagonalized in the usual way based on Gaussian elimination, and then count the number of positive and negative elements. Since using arbitrary precision in the process of Gaussian elimination, even when using exact division (see [F. Rouillier 94] and [H. Hong, W. Stöcher 94]), causes exponential growth of the matrix elements we first try to compute the diagonal form over double float intervals. As for computing the signature we are only interested in the signs of the resulting matrix this method often suffices, especially when given a matrix of full rank. If the signs cannot be determined uniquely the computation with arbitrary precision is started.

7 Complexity

Here we give the order of complexity of *only parts* of the algorithm in terms of multiplications. We do not consider here that the complexity for arbitrary precision arithmetic heavily depends on the length of the numbers involved. A full complexity analysis, for instance for reduction w.r.t. a Gröbner basis goes beyond the scope of this report.

Constructing the basis
Starting with 1 for each basis element constructed at most r neighbours have to be checked as the next basis element, each checking having a cost of $O(r)$. Hence the total cost is $O(dr^2)$, with a small factor, though.

Computing the Multiplication Table
In the worst case there are $\approx d^3/2$ elements to compute, in general only $O(d^2)$. The computation of each element involves the reduction of a product of two basis elements w.r.t a Gröbner basis, which is hard to estimate.

Computing the Quadratic Forms
Each quadratic form involves the computation of the vector $t_k^{<h>}$, and based on this vector the elements. Computation of the vector involves $O(d^2)$ multiplications. In the worst case there are $\approx d^2/2$ distinct elements in an associated quadratic form, $O(d)$ in the general case. The computation of each element involves d multiplications, giving $O(d^3)$ multiplications all together in the worst case, but only $O(d^2)$ in the general case.

Computing the Signatures
Computing the signatures is based on Gaussian elimination, involving $O(d^3)$ multiplications.

Table 2: Timings for intermediate steps in ms

Data	w	Q	$Q^{<h>}, Q^{<h^2>}$
Example	830	90	480

8 Experiments

Let us consider the example consisting of the following polynomials:

$$f_1(x,y) = x^5 - x^2y^2 + y^3 - 1,$$

$$f_1(x,y) = x^3 - xy^2 + y^2 - y + 1,$$

$$h(x,y) = 1 - x^2 - y^2.$$

The timings for computing the intermediate steps are in table 2.

References

[BKR 86] M. Ben-Or, D. Kozen, J. Reif *The complexity of elementary algebra and geometry*, J. of Computation and System Sciences 32 251-264 (1986).

[D. Cox, et al. 92] D. Cox, J. Little, D. O'Shea *Ideals, Varieties and Algorithms - An Introduction to Computational Algebraic Geometry and Commutative Algebra*, Springer New York (1992).

[P. Pederson, et al. 94] P. Pederson, M.-F. Roy, A.Szpirglas *Counting Real Zeros in the multivariate case*, published in ?.

[F. Rouillier 94] F. Rouillier *An extension of Bareiss method to the reduction of quadratic forms*, report in the POSSO project, Universite de Rennes, France.

[Hong et al. 94] H. Hong, A. Neubacher, V. Stahl. *The STURM Library Manual - A C++ Library for Symbolic Computation* Technical Report 94-30, RISC Linz, J. Kepler University, Linz, Austria, 1994.

[H. Hong, W. Stöcher 94] H. Hong, W. Stöcher *Exact Division Applied to Diagonalizing Symmetric Matrices*, RISC Report 94-35, RISC Linz, J. Kepler University Linz, Austria.

[H. Hong et al. 95] H. Hong, W. Stöcher *Structure and Efficient Computation of Multiplication Tables and Associated Quadratic Forms* , RISC-Linz Report Series No. 95-51, RISC Linz, J. Kepler University Linz, Austria.